Advanced Java and Cutting-edge Applications

Project description

The Context

You work at a startup that develops software and computer applications. You decide to respond to a request for proposals (RFP) from a large company, which we'll call **BigComp**. To respond to the RFP and try to win the contract, you must produce a prototype that you will demonstrate to managers and department heads within the company, who will then select the best software. The prototype you create must be as close as possible to the final application and must simulate the parts that require physical deployment in the application's real-world environment. In particular, the graphical user interface must be particularly polished and as close as possible to its final version. Similarly, the database structure must be almost identical to the final structure.

General Specifications

BigComp wishes to implement a comprehensive access control system to secure its headquarters, which consists of a large property comprising several buildings, each with multiple floors, parking lots, etc. For security reasons, the control system must be a native application and cannot be a web application. The company wants to secure all *physical* access points within its site:

- site entry and exit, on foot or by vehicle
- building entry and exit
- access to doors leading to offices, meeting rooms, corridors, technical rooms, the data center, etc.
- elevator and stairways access to different floors
- access to underground parking

The company also wishes to use the same system to control access to *resources* available on site, primarily:

- the use of printers
- the use of beverage dispensers (free for employees)

To simplify the project description, the word "resource" will be used to refer to both gate-like resources (doors, gates) and other resources (elevator button, printer, beverage dispensers). After studying the solutions available on the market, the company opted for a *badge-based access control system*. An access badge is a physical or virtual (smartphone) identification device issued to an authorized person (company employee, but also contractor, intern, visitor) to allow them access to secure areas or resources. It contains a unique identifier (a sequence of characters) that the access control system reads and verifies to authorize (or deny) access to the resource (opening a door, a gate, etc.). The company informally specifies the general characteristics expected of the access control system:

- the system must be highly configurable, allowing the creation of standard profiles (employee, project manager, intern, contractor, etc.) that can be used to quickly configure a badge, all through a user-friendly graphical interface
- the system must offer the ability to save these profiles and make them persistent, so they can be reloaded into the system in case of a shutdown and restart
- the system must allow for the very rapid invalidation of a badge
- the system must offer a complete graphical user interface to interact in all ways with the access control system
- the system must offer a graphical interface for visually monitoring the activity of the various access points and resources on the site. For example, it must be possible to view the resources on a floor of a building, as well as access to those resources in real time
- the system must maintain daily reports on all accesses (granted and denied access attempts) and record all relevant information (badge ID, resource ID, badge holder name, date, time of access, etc.) for each of them
- the daily reports (a.k.a. logs) must be easily accessible through a graphical user interface that allows for relevant searching of information (for example, searching for all accesses to a given resource within a given time period)
- the system must be flexible and scalable, allowing for the future addition of new resources (such as electric scooters available on site for employees to move quickly between buildings)

Regarding resource access, the company requires a high level of configuration and control. Access conditions to a resource depend on a combination of several factors, including:

- the user: this is the primary criterion. A user is generally associated with a specific profile that defines their access rights
- the date: access rules for the resource may depend on the date (year, month, day, hour, minute). For example, a door may only be able to open during office hours

- the number of accesses: some resources may have a limited number of accesses (per day, per week, per month). For example, beverage dispensers provided free of charge to staff are only accessible no more than N times per day and per person
- precedence: the access control system takes into account certain precedence in the use of a badge. For example, a badge cannot open a door inside a given building if that same badge has not previously opened (within a given timeframe) an access door to that building (this mean a person would have entered a building without using a badge, which is prohibited)

The access rights definition system must be highly flexible and configurable, as **BigComp** can add new criteria at any time. Notice that the access conditions appearing in red are optional, and must implemented as a bonus (e.g. you must focus first on the essential features of your prototype).

Simulation

The application requested by **BigComp** includes a "physical" part (the badge readers and their connection to a network, and the resources they control) and a purely software part (the part that will process access requests, view them, record them, etc.). To participate in the tender, you must therefore develop a **complete prototype** of the application, which must include three parts:

- a software component: this is the core of the application, handling request processing, monitoring, and database management, all through a graphical user interface
- a "physical" component that must simulate badges, badge readers, the resources they control, and the connection between the badge reader and the access control system
- an "events" component that must simulate the people using the system in real time

Badge Simulation

A badge is simulated by a Java object containing the information stored on the badge. This information is only readable by a badge reader. A badge can be reprogrammed, but it is only re-programmable by a badge reader. For security reasons, the information stored on a badge is limited to a code. This code identifies a unique user and is used by the access control system to decide whether access to the resource is authorized. The system has the ability to reconfigure a badge, that is, to modify the information stored on it. A badge has an expiration date, and once that date has passed, it no longer grants access to any resources. The system allows to update a badge and set a new expiration date. Notice that, for security reason, the expiration date (or any other information) is **not** stored on the badge.

Badge Reader Simulation

A badge reader is simulated by a Java object that can read the information (the code) stored on a badge and transmit it over a network. A badge reader is associated with a resource (for example, a door) and can make the resource accessible or not (for example, by opening the door). When a user swipes his badge on a badge reader, a message appears on the badger reader (like "access granted", "access denied", etc.). A badge reader can also modify the information stored on a badge (the code). This capability enhances system security: the codes of all badges can be changed periodically to prevent any fraudulent use of a stolen or copied badge. This process should be automated: when it's time to update its code, any time the badge is swiped on a badge reader, a suitable message should be displayed (like "badge must be updated"). The user has then a limited period of time to update it. If the badge is not updated within that period, the badge becomes non functional, and any swipe on any badge reader will result on an access deny with a suitable message (like "badge not valid"). To update the code on a badge, the user must hold the badge up to the badge reader (and not swipe it) during the time it is upgraded. This means your badge reader object supports the two actions:

- swipe a badge
- update a badge

In case it is not the time to update the bade, holding it up to the badge reader does not produce any effect.

Resource Simulation

A resource is a Java object whose attributes describe the resource's characteristics. For example, a resource might be a door, and its attributes include its (unique) identifier, its physical location on the site (building, floor, etc.) and its state. A resource can have two states:

- controlled state: the user must swipe their badge on the reader, and if they have the appropriate access rights, the resource becomes available (for example, the door opens).
- uncontrolled state: the resource is available (or not) without verification (for example, the door is always open or always closed). In this case, the badge has no effect.

It must be possible to change the state of one or more resources. For example, the application should allow, with a single click (from the control interface), to set the state of a number of doors (selected in advance) to "uncontrolled" to facilitate building evacuation in case of fire. This feature can be implemented using *groups* of resources (see the section *Handling profiles*).

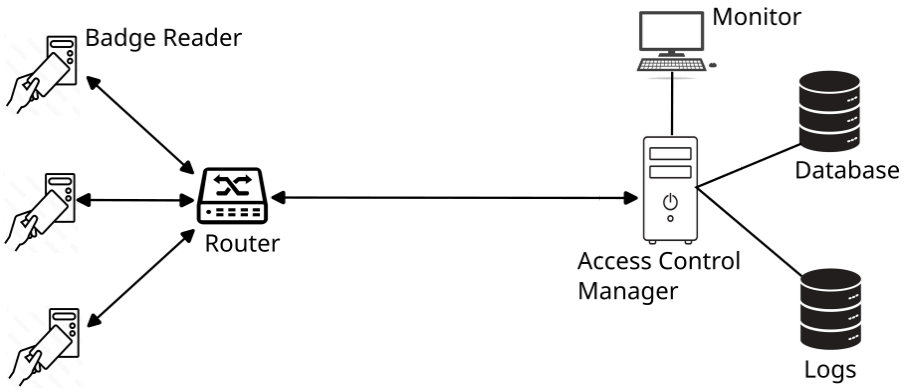
Events Simulation

To ensure your prototype is convincing, you must simulate the use of your application in a real-world situation. To do this, you must simulate the activity of (several) people with badges using the system. You must be able to define users and simulate their activity (using their badge to access resources) consistently. For example, a simulated user should perform actions that are consistent in

time and space (entering through the main entrance first, then entering a building, and finally going to an office within that building). To demonstrate the possibility of date-based access restrictions, you must be able to modify the current (absolute) time to test different scenarios. For example, if access to certain resources is not permitted on Sundays, you must be able to set the current date to a new date corresponding to a Sunday to verify that access is indeed prohibited. Your events simulation must be scalable: your system should work with a realistic number of events, something like 300 badges (users) and 400 badge readers (resources).

Handling an Access Request

The access control system architecture can be summarized by the following figure:



As mentioned above, a resource can be in a *controlled* or *uncontrolled* state. A resource in *uncontrolled* state is immediately available and renders the badge reader inactive (e.g., an *uncontrolled* door does not use the badge reader to open). The complete processing of an access request to a controlled resource is detailed here:

- a user wishing to access the resource (for example, a door) swipes their badge in front of the resource's badge reader.
- the badge reader reads the code on the badge and generates a request consisting of:
  - the code on the badge: this code uniquely identifies the badge owner.
  - the badge reader's code: this code uniquely identifies both the badge reader and the resource to which it is associated.
- this request is sent to a router, which in turn forwards it to the control system. Throughout this process, the badge reader is **not active** (e.g. any badge swipe has no effect)
- the control system processes the request: the request is faced with various filters that depend on the user (via their code).
- if access is authorized, the control system sends a message back to the router containing:
  - the badge reader code
  - a special value indicating that access is authorized.
  - upon receiving this message, the router forwards it to the correct badge reader. Upon receiving the message, the reader grants access to the resource (for example, opens the door). After a predefined (short) period, the resource becomes unavailable again (the door closes).
- if access is not authorized, the control system sends a message back to the router containing:
  - the badge reader code
  - a special value indicating that access is not authorized.
  - upon receiving this message, the router forwards it to the correct badge reader. Upon receiving the message, the system displays an alert indicating that access to the resource is denied.
- at the end, the badge reader becomes **active again** (and therefore badges can be swiped again).

To closely mimic the operation of a physical resource, you can use *virtual threads* to simulate the resource's operation time. For example, opening and closing a door takes a few seconds during which the badge reader controlling the door is inactive. Concerning the connections between the various components, you don't have to simulate them by implementing actual connections between different Java processes. For example, the router can be just a part of the main core and you don't have to make it a remote component. The connections between the badge readers and the router can be implemented using the *PropertyChangeListener* like we did in the lab 8, but you are free to use another implementation.

Checking access

When a user wishes to access to a resource, an access request is sent to the core of the system, the *access request processor* (ARP in short). The response time of the ARP must be as short as possible to grant or deny the access in real (and short) time. Therefore, the access rights information must be loaded in the live memory and the ARP should **not** require any access to the database to perform the checking of the access request. This implies that at startup, the system must upload the access rights information described through profiles (see the next section) stored in files. These information must be organized into the suitable data structure to optimize the matching of the access request. This data structure must combine both running time and memory usage efficiency. For example, a two-dimensional matrix with badge IDs as rows and badge reader IDs as columns (or vice-versa) may not be optimal in terms of memory usage.

Handling profiles

The company **BigComp** requires that the application provides a way to define *profiles*. A profile defines the access rights for a user. Each user is assigned one or more profiles. The resources controlled by the access control system are organized in groups of different security levels. For example, the main gates to enter the company domain are in a group of low security level (e.g. any authorized user should at least be allowed to enter the domain). On the opposite, the doors to access the computing center are in a group of high security level. An access right is a pair *(group, time filter)* which defines the *time* the resources from the group are accessible. A profile is a list of access rights. It must be possible to define the time when access is allowed in several ways:

- by listing a series of year, months, days of the month, days of the week or hours
- by an interval of year, months, days of the month, days of the week or hours
- with the opposite of the previous ones
- with any combination of the above

Here are some examples of possible time filters:

- 2025. July,August. Monday-Friday. 8:00-12:00,14:00-17:00 : using this time filter, the resource is accessible, for the year 2025, in July and August, Monday to Friday, from 8:00 to 12:00, and from 14:00 to 17:00
- 2026. EXCEPT June, July, August. EXCEPT Sunday. ALL : using this time filter, the resource is accessible, for the year 2026, each month except June, July and August, all week days except Sunday, at all time
- ALL. ALL. Monday-Friday. EXCEPT 12:00-14:00 : using this time filter, the resource is accessible any year, any month, from Monday to Friday, any time except from 12:00 to 14:00
- 2026.January,March,May. Monday,Wednesday,Friday. 16:00-18:00 : using this time filter, the resource is accessible in 2026, in January, March or May, on Monday, Wednesday or Friday, between 16:00 and 18:00

You can create your own time filters, and you need to find **the optimal way to implement them in Java** so that matching with a date is as efficient as possible. To make them persistent, the profiles must be stored in files, which requires that also the groups must be stored in files. You have to define a syntax for:

- group files: a group file is storing some resources. You may store in the file all the relevant information about a resource (name of the resource, location, ID of the resource, ID of the badge reader controlling the resource). The name of the file should match the name of the group it is storing.
- profile files: a profile file contains a list of pairs *(group, time filter)*. You then have to design a syntax to encode time filter, and you can get inspiration from the above examples of time filters.

A group can be very small or quite large. For example, a permanent employee may be authorized to access his office at all time, and in that case would have the profile made of:

- a group containing only the door (the resource) of his office
- the time filter ALL.ALL.ALL.ALL

When deciding about the file format to store profiles, remember that **the system will have to upload those profiles from their file representation to the live memory**. So you should design first the Java implementation of how the profiles (e.g. the groups and the time filters) are going to be used in memory, and then decide about their string representation in files. A standard solution is to use XML or JSON, but you can use your own text representation. Finally, although it would be expected to be able to create full profiles from a graphical user interface, you may skip this part and and only provide the part to create groups of resources.

The log system

All access requests sent by badge readers must be recorded by the logging system. It should be possible to search the logged data using dynamic filters that the user can select through an appropriate graphical interface. To make it more convenient and more flexible, the logging system will store the access requests is CSV format. For example, a logged (granted) access request could look like:

```
2025,Dec,24,Wed,14:36:49,BX76Z541,BR59KA87,R7U39PL2,83746028,John:Doe,GRANTED
```

where:

- the values *2025,Dec,24,Wed,14:36:49* store the date and time of the access request
- the value *BX76Z541* is the badge code
- the value *BR59KA87* is the badge reader code
- the value *R7U39PL2* is the identifier of the resource controlled by the badge reader
- the value *83746028* is the identifier of the owner of the badge
- the value *John:Doe* is the first name and last name of the owner of the badge
- the value *GRANTED* indicates that the access was granted (*DENIED* if the access was denied)

To facilitate their searching, the log (CVS) files, could be stored in a structured way:

- one log (CSV) file per day
- all the log files for a given month in the same folder (one folder per month)

- all the month folders for a given year in the same folder (one folder per year)

To be able to demonstrate your system's ability to search through logs, you need to artificially create several log files for different days and different months.

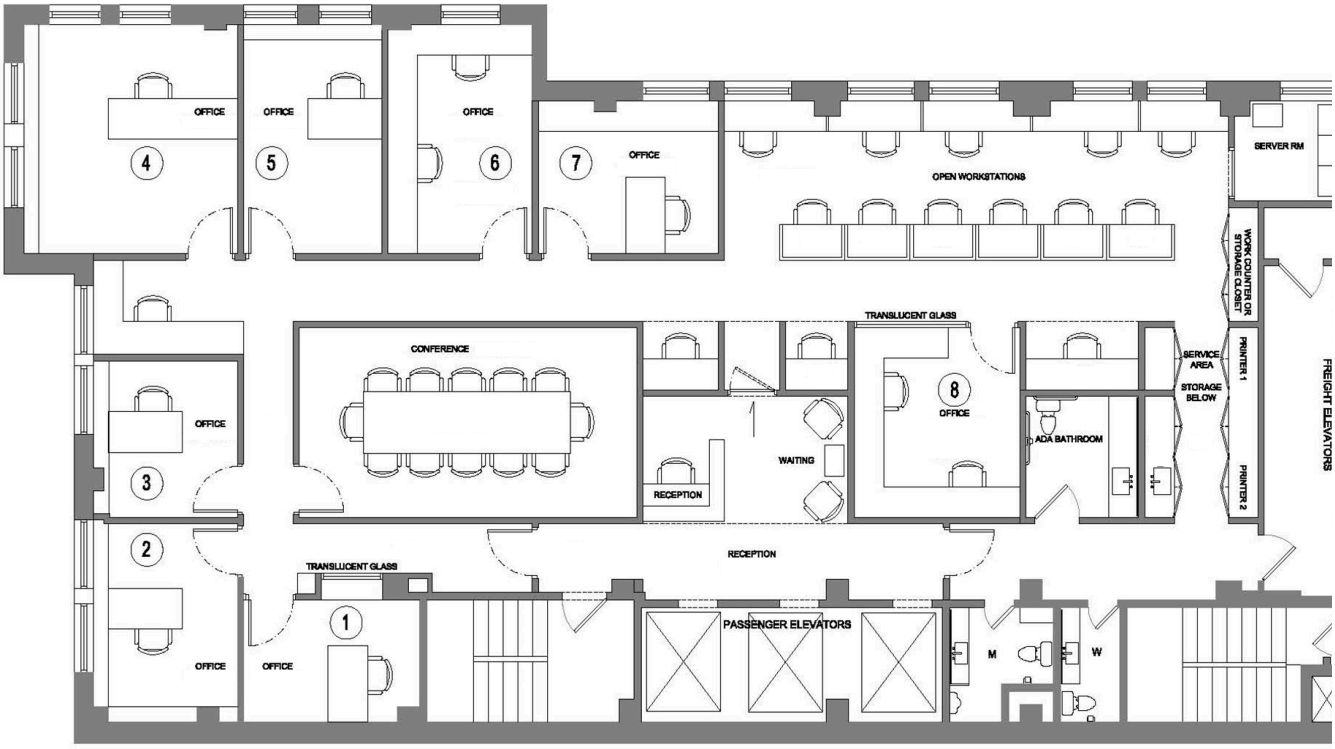
The User Interface

The access control system must provide a **graphical user interface** (GUI) for configuring and monitoring the system. In particular, this interface must allow users to:

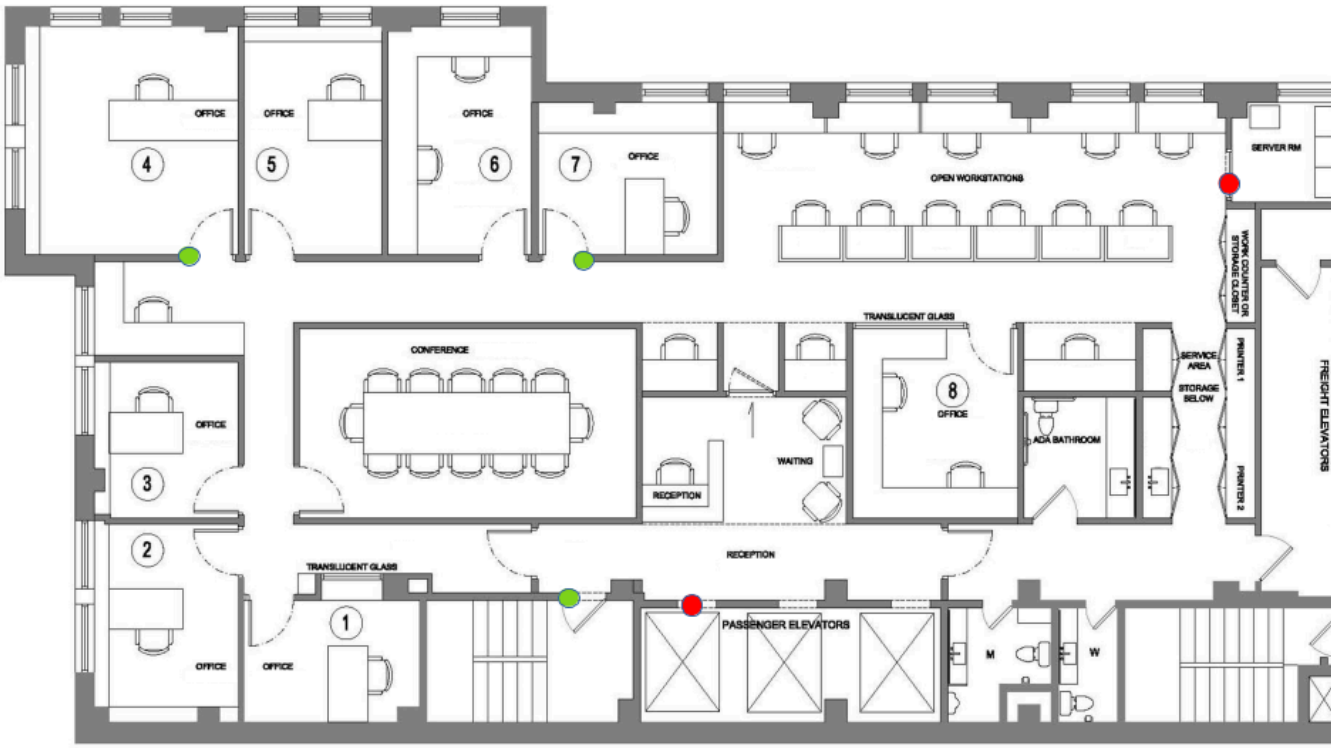
- register or delete a new user: the system must support different types of users (permanent employees, contractors, interns, visitors, etc.)
- define, modify, delete, save or restore user profiles, allowing for quick and easy assignment of access permissions to users: for each user, the profile defining their access rights can be modified
- view daily reports: the interface must offer a search function with filters to select entries in the daily reports based on various criteria
- visualize all or part of the site and show access attempts in real time: the application must allow users to visualize parts of the physical environment (for example, the site plan or a specific floor of a building) and display access attempts to different resources in real time

For site visualization, since you are developing a prototype, you can limit yourself to **two** visualization examples: a general site plan and a specific floor of a particular building. You can optionally use the following image to visualize the site:

You can upgrade this image, for example by adding names on building, and by adding gates at the entrance of the site. The following image can be used to visualize a building floor:



The access control system must visualize attempts to access resources (doors, elevators, etc.) by displaying, for example, a flashing green dot at the location of the badge reader when the request is accepted, or a flashing red dot when the request is denied :



The database

The access control system must handle a *database* to store various information, including:

- the *users* of the system: the attributes of a user include the gender, the first and last names, the ID number, a reference (foreign key) to his badge
- the *badges*: the attributes of a badge include the ID of the badge, a reference (foreign key) to its owner, the date of creation, the expiration date, the date of the last update
- the *resources* controlled by the system: the attributes of a resource include the ID of the resource, a reference (foreign key) to its badge reader, the name of the resource, its location, the state of the resource (CONTROLLED or UNCONTROLLED)
- the *badge readers*: the attributes of a badge reader include the ID of the badge reader, a reference (foreign key) to its resource
- the *groups* of resources: the attributes of a group of resources include the name of the group (primary key), some kind of security level (like an integer), a reference to the file path defining the group

- the *profiles*: the attributes of a profile include the name of the profile (primary key), a reference to the file path defining the profile

Besides those tables, you need to include some *junction* tables, for example between *badges* and *profiles*: a *badge* must be linked to multiple profiles, and a *profile* can be one profile of multiple badges. You may think about more information to store and more attributes than the ones listed before.

Supporting files

The supporting files are:

- site-layout.png: this is the image of the company domain
- office-layout.png: this is the image of a floor of a building

You may use other images in your project.

Last modified: Thursday, 25 December 2025, 12:20 PM

◄ Project how-to

Jump to...

Step 1 upload ►